Towards an Interactive High Visual

Complexity Animation System*

Ъy

C. Csuri, R. Hackathorn, R. Parent, W. Carlson and M. Howard

Computer Graphics Research Group

The Ohio State University

ABSTRACT

A computer animation system is discussed which employs interactive techniques and presents a unified approach to the graphical display of complex three dimensional data. The system facilitates the generation, manipulation and display of highly detailed data with the aid of interactive devices and a video interface to a standard color TV monitor. The system enables the animator to create a variety of objects (including texture) and to specify the necessary transformations for animation sequences. A run length processing technique combined with a brute force Z-buffer algorithm has been newly designed and implemented that can handle the intersection of several million faces, lines and points. This makes possible a full range of visual cues to simulate fire, smoke, water and complex 3-D texture such as grass, hair and bark. Basic concepts and approaches are described. The display algorithm and the procedure model to generate texture are presented and the implications of the system for computer animation are discussed. Extensions to the system are outlined which include a unique graphics display processor currently under construction that includes a partial implementation of the display algorithm in hardware.

INTERACTIVE SYSTEMS AND HIGH VISUAL COMPLEXITY

Computer animation has advanced to the stage where systems have been implemented which can routinely handle the 3-D display of objects or scenes involving several thousand faces (6, 7, 10). We now need to develop interactive systems that can facilitate the generation and display of more complex data. This is especially true for representing detail such as textures of hair, feathers, grass or the display of smoke, fire and liquids. Early work in this area demonstrated that computers can generate and display pictures requiring several hundred thousand polygons or surface patches. These pictures were developed for the most part in the context of a batch processing environment. MAGI's unique algorithm produced a color picture in 1974 (2) of a deciduous tree that had several thousand leaves, Using a procedure model to generate or "grow" the tree, it took three hours of calculation time on

*This work is supported in part by National Science Foundation Grant Number MCS76-18659. an IBM 360/65 computer to display this image at 500 x 500 resolution. Crow (5), with a visible surface algorithm and a technique of object instancing that he developed for Information International Inc., displayed pictures consisting of several hundred thousand polygons. His picture of multiple copies of the ABC logo took over 30 minutes of calculation time on a PDP-10 with a KA-10 processor. The emphasis in his system is on image quality and he uses a 3000 line display for producing pictures which represent the state of the art in computer graphics. Catmull (3) produced complex pictures of bottles and glasses (142 bottles and glasses where each bottle required 32 bi-cubic patches). Newell (9) also displayed high complexity pictures; the most notable of which is entitled "Pawns" in which he used instancing to generate the data and a display processor that incorporated a hardware implementation of the Watkins algorithm. Using a PDP-10 computer and this graphics processor the picture took approximately twenty minutes of calculation time.

© 1979 ACM O-89791-004-4/79/0800-289 \$00.75 See Copyright Pg.

All of these efforts were important steps towards high visual complexity animation systems. Even though we now have improved technology which helps in some aspects of the display problem, there is more work to be done with interactive techniques and color raster scan displays, data generation (especially for images such as texture and smoke), the specification of complex creature motion, display algorithms and image quality. We must find methods that will enable us to easily generate, manipulate and display the intersection of several million faces, lines and points in three dimensional space.

Although there are interactive two and two and one half dimensional systems that employ a TV color display (12), there are not many examples of interactive 3-D systems. Jones (7) implemented an interactive black and white system with extensions to ALGOL-60 in Case Western Reserve's high performance graphics display processor built by the Evans and Sutherland Corporation. It is an animation system that handles several thousand edges in real time. An interactive 3-D graphics system involving TV display was developed by Staudhammer (10) at North Carolina State University which refreshes a 3-D image from an analog disk. Other systems have been built for the military, such as flight simulators developed by E and S and G. E. Corporation. These are special purpose systems built at great cost and designed exclusively for flight training that have many of the interactive features required for an animation system. While they perform well for their intended purpose, there is a limitation as to the amount of data that can be displayed in real time; this is usually around 3000-3600 edges. Although better hardware implementations are possible, the display algorithms do not seem well suited for high complexity data. There has been some interesting exploratory work by Clark (4) that may overcome some limitations. His techniques involve a hierarchical organization of large data bases that has significant implications for highly complex data and the visible surface calculation. In his scheme the requirements for the display of detail at any instance in time is, in part, determined by one's viewing position in 3-D space or one's position within a tree structure. Unnecessary edges are not processed by the algorithm, thus reducing the calculation time.

ANIMA II (6), which was one of the first attempts at an interactive 3-D color raster scan animation system was designed and implemented by the CGRG at the Ohio State University. It enables the user to generate, manipulate and display 3-D images and it has a real-time playback capability for animation. The animation sequences can be recorded onto videotape. This system has been used to produce over two hours of storyboard animation for science, education and commerical TV. The strengths of the system include its interactive capabilities with the data generation subsystem and the relatively easy to use director's language. Another attractive feature is a fast visible surface algorithm for the rapid display of animation which provides users with the spon-

taneity necessary to make visual judgments. The run-length encoding scheme used as final output to a video interface designed for us by Staudhammer makes this particular combination of software and hardware work for the real-time playback of animation. There are limitations to the ANIMA II system that are primarily related to image quality. The data transfer rate from disk of the playback scheme limits the number of runlengths available per frame which affects the image quality attainable. While complex creature motion has been created with the system. the specification of transformations, especially the positioning of sub-parts, tends to be time consuming. There is no smooth shading, transparency, highlights or shadows and aliasing is a problem. It should be noted that an earlier version of the Myers' (8) Z-buffer algorithm implemented in 1974 at CGRG handled smooth shading and transparency, but that these features were eliminated because greater value was placed upon user interaction and a real-time playback capability.

The primary focus of this paper will be upon our new animation system which seems to grow naturally out of our previous work with ANIMA II. A later unpublished version of the Myers' algorithm (an efficient implementation of a bruteforce Z-buffer algorithm) was designed to process data directly off a mass storage device. While Myers' work has produced good preliminary results, we have found it to have certain limitations. The type of internal data structures used and the requirements of separate copies of data limits the speed of calculation and the variety of visual results that can be displayed. Recently, Hackathorn of CGRG has extended this previous work by introducing features and techniques that represent a modest improvement for the display of high complexity data. The display algorithm incorporates a unified approach to the display problem where one can combine not only polygonal surfaces but also points, lines and bi-cubic patches.

A new animation system called ANTTS has been implemented on our PDP-11/45. It is an experimental system with a language that involves interactive techniques and a unified approach to the display of high complexity data including textured objects and smoke. This system is being redesigned for implementation on our VAX/780 computer with a unique graphics display processor under construction by the CGRG. The final system will exploit the display algorithm with a partial implementation in hardware and it should significantly improve response time for interactive capabilities.

DESIGN CONSIDERATIONS

An interactive high visual complexity animation system should have several basic features:

1. Ideally it should be a real-time system, but with available technology and associated cost an interactive capability seems like a reasonable goal.

2. A language is required to specify the transformations to control the object's

position and movement through space. There is a need for a facility where the moving human figure can be digitized and a motion file created.

3. Techniques such as procedure models are necessary to generate and store complex data. Data generation could be an algorithm to "grow" a tree, instancing or solids of revolution, including the triangulation between slices. One should be able to specify different resolutions for the object, for example, displaying an object having up to one million faces as a simplified object having only ten faces. This is important if one hopes to achieve an interactive design capability.

4. The system should have facilities for the digital editing of animation sequences from a mass storage device. It should allow for the random access of sequences and the dynamic changing of color in real-time playback mode without recalculating the sequences.

5. The display algorithm should be able to handle the intersection of lines, points and surfaces in three dimensional space. Data should be processed in a stream (as opposed to pre-sorting), either by reading in data directly from a disk or by creating data "on the fly."

6. The display algorithm should lend itself to a simple hardware design. A hardware implementation of the entire display algorithm, or at least an implementation in microcode, would overcome computational constraints. The level of effort in hardware seems to be more a function of what one can afford rather than what one can conceptualize. Anti-aliasing is part of the display problem and it seems like the real solution belongs in the realm of hardware.

VISIBLE SURFACE ALGORITHM

In general we can characterize two classes of visible surface algorithms: those which directly calculate all visible surfaces keeping them intact entities, and those algorithms in which the visible surfaces are indirectly found as a result of calculating the image at the pixel level. To the first class belong the subdivision algorithms of Weiler and Warnock that calculate the visible polygons (or polygonal areas) in a surface and which must be separately converted to a raster scan format. Also in this first class are the scan line algorithms such as those designed by Romney, Bouknight and Watkins. These algorithms directly determine a visible surface in terms of its visible segments (in image space) and each of them uses raster scan conversion as an integral process. All algorithms of the first class have one characteristic in common: they have at least one nonlinear sorting step which makes them seem inappropriate for images with more than about 10,000 polygons. Several of these algorithms use some form of coherence to "hedge" against the inevitable calculation explosion, but few coherence schemes work well with the varied

and detailed data descriptions in a complex scene (with the possible exception of pixel coherence run lengths).

The algorithms in the second class are typified by the use of a frame buffer to indirectly determine which surfaces are visible. This technique makes them more suitable for the processing of high complexity image data. A frame buffer, usually used for video display, is nothing more than a two dimensional array of memory that stores all the picture information necessary to describe one complete frame. Frame buffers can be distinguished by whether or not they allow a full 2-dimensional bucket sort in both 'X' and 'Y', of if they only allow a 1-dimensional bucket sort in just 'Y'. This last type of frame buffer is called a run-length frame buffer and it uses a fixed length block of memory at each scan line in order to hold a list of run lengths which are visible on that scan line. The other frame buffer type is usually organized as 512 rows (scan lines) with each row containing 512 fixed blocks for pixel (picture element) storage. This type of frame buffer is characterized by the size of the pixel blocks, i.e., how much information each pixel contains. The simplest of this type is called a <u>2-D color frame buffer</u> and has either 1, 4, or 8 bits per pixel which indicate the pixel's color (often by pointing into a palette look-up table). The type known as a Z frame buffer or depth buffer is a more sophisticated one. Here, enough bits to adequately store the pixel's position along the 'Z' axis is kept along with the color bits for each pixel. This enables brute force comparison techniques to retain only the pixels closest to the observer. A third type of frame buffer in this categorization scheme is what we call a pixel buffer. A pixel buffer not only has the ability to be a 2-D color frame buffer or a Z-buffer, but it can also hold additional information (all optional) about the pixel. This could include: original object identification number, object geometric type (curved, planar, line, or point), object image type (solid, transparent, shadowed, etc.), positional pixel skewing in 'X' and 'Y', pixel transmittance, pixel reflectance, and additional color information which could be used for transparency or shadows.

Data is moved into the frame buffers in one of two methods, either unconditional overwrite or conditional overwrite. Newell is the best known advocate of unconditional overwriting and he has written two such algorithms. Both his algorithms depth sort a list of polygons (dividing where overlapping occurs) and then simply write the polygon into a frame buffer starting with the most distant face from the observer. His first algorithm used a run-length frame buffer, but this was abandoned for a simple 2-D frame buffer in his second algorithm because of the extra time spent ordering (merging) the runlengths in 'X', and because the number of runlengths associated with each scan line severely limits the image complexity.

Conditional overwriting is found with the frame buffers which store 'Z' information at each pixel. Here a brute force comparison is made at a new pixel's 'Z' position, and that pixel is overwritten into the frame buffer only if it is closer to the observer than whatever was in there previously, or else it is ignored. Catmull, Myers and Crow have used the conditional overwrite approach as the heart of their visible surface algorithm.

A variation of unconditional and conditional overwrite is unconditional and conditional modify. Again, these techniques are used with frame buffers but instead of a direct replacement, a third pixel is formed as a result of two pixels (the current one in the frame buffer and the new pixel) combining and modifying each other. This condition is common when the effect of transparency is desired.

When one deals with very high complexity data, a consideration that becomes almost as important to a display algorithm as its visible surface algorithm, is the manner in which the data is arranged and presented to the visible surface routines. There are two basic approaches. One approach converts an incoming list of object space data into one or several other lists of image space data in an attempt to make the data more "digestible" by the visible surface routines, The other approach is to treat each patch, face, line or point of the incoming list as an isolated element, where the surface routines work directly on it in a "stream processing" manner.

THE DISPLAY ALGORITHM

R. Hackathorn of our group has approached this problem by choosing to process data in a stream, either by reading in data directly from

a disk or by creating data "on the fly" with procedural algorithms. In order to do this stream style of processing, his display algorithm relies heavily on two large memory arrays called a runlength buffer and pixel buffer (frame buffer), respectively. The run-length buffer always exists in main memory while the pixel buffer is always kept in disk memory with large blocks brought in and out of main memory as needed.

Our choice to internally represent all data as run lengths is not only a very efficient and fast method to store data, it also allows for a unified aproach to the display of data. In this process, triangles and lines break down into run lengths, and points and patch points are merely run lengths of length 1. This means that all these data types can exist together naturally within a scene description without the need to deal with special cases (see Figure 1).

While the display algorithm is very simple (i.e., run-lengths buffer to 'XY' pixel buffer using brute force 'Z' comparisons), it has traded off capabilities, primarily the ability to compare a face against other faces before making a determination of their visible parts. This has traditionally been required for such features as shadows, transparency, reflections and antialiasing. However, solutions exist for these cases, and though they may not yield the most optimal results, their output is a close enough approximation to be useful.

Since animation in a video environment requires thirty frames per second, it is only natural to want as little information in the test



FIG.I

frames as is feasible. To achieve variable complexity, an object consists of a number of different representations. At the lowest level, an object can exist as a skeleton made up of points and lines. A final complexity level can consist of smooth, curved or textured surfaces mapped onto the skeletons' positions. Variable image quality can range from very low resolution (64 x 64) up to high resolution (1024 x 1024 averaged to 512 x 512) and include options for image enhancements.

FLOW OF PROCESSING FOR THE DISPLAY ALGORITHM

Assuming polygons as the data type, the flow of processing is as follows:

- Get a single triangle from disk or generate one using a procedure model algorithm.
- Update the triangle's 3 points with position, rotation and size transformations.
- Determine face color from angle of incidence between the light source, triangle and observer.
- Convert the triangle's points from an object space coordinate system to one of image space using a perspective projection.
- Orient the triangle definition with respect to its highest 'Y' value.
- Raster-scan convert the triangle into run-lengths with one for each scan line the face crosses.
- 7) Pass each run length to the buffer by adding it to the end of an unordered list of previous run-lengths found at the same Y scan line.
- 8) When all triangles have been processed for one frame, the run lengths are decomposed into 3-D pixels and have their 'Z' values compared against a pixel with the same 'XY' values in the pixel buffer, effecting brute force hidden surface removal. The pixel buffer is then read into main memory (one section at a time) and encoded into video run lengths for display on a TV.

ADVANTAGES OF THE DISPLAY ALGORITHM

The combination of a run length buffer and 'XY' pixel buffer eliminates three common "growth pains" found in algorithms processing over 100,000 polygons (or the equivalent with patch algorithms):

 There are no non-linear sorts by doing a 'Y' bucket dump into the run length and, when needed, doing a combination 'X' bucket dump and brute force 'Z' comparison into the pixel buffer.

- 2) There are no internal image space datalists which grow in length with the number of polygons. Variable image lists such as active faces/ edges per scan line or depth ordered faces or the dual list output of subdivision algorithms must be stored in disk memory when the image complexity gets too high. The use of disk storage distracts from the advantages of procedural model data generation, and competes with disk-resident object data descriptions for space, thereby limiting maximum complexity.
- 3) The use of a run-length buffer virtually eliminates the timecosts in randomly accessing a diskresident 'XY' pixel buffer by spreading the cost of disk transfers among many pixels. For example, if each scan line in the run length buffer holds 100 run lengths and 20 scan lines are moved to the pixel buffer when a scan line is full, and if each run-length describes an average of 3 pixels, then there is a potential of up to 6,000 pixels involved with each disk access. This translates into a cost per pixel of only a few micro-seconds for each memory access.

Another advantage of this combination is that it provides a simple and flexible approach to high complexity image synthesis. This allows the display algorithm to operate in a variety of modes and interface to a variety of external programming tasks. By using run lengths as the primitive data type, an internal image space data structure can be built which is common not only to simple images such as lines and large planar polygons, but also to complex imagery made of surface patches or millions of small triangles or points. Such a feature greatly facilitates interactiveness in an animation environment by allowing objects to be seen at various levels of detail.

While the use of a run length buffer provides a common (fixed size) internal data structure, using a disk resident 'XY' pixel buffer provides a common sharable image. This makes possible a more flexible approach to generating video images than we have used in the past. Using our run length encoded video interface for output we have written a 2-D color "painting" routine, a "star generation" routine (for producing interacting galaxies), and various random background generation routines. But none of these programs could mix their image output with that produced by the display algorithm of ANIMA II, nor could ANIMA II mix in the output of any of these routines. By using a disk resident sharable image such as that found in the 'XY' pixel buffer, these incompatability problems are overcome simply by storing a common disk resident directory and status block, and allowing various routines to access

any one of several 'XY' pixel buffers using techniques of conditional or unconditional overwrite or modify overwrite.

This approach makes possible a variety of useful interactive animation techniques as well as post processing techniques for special effects and image enhancement. One interactive technique is to store several pixel buffers on a disk, but use just one to process the video image. At the start of each frame, the separate pixel buffer can either be cleared (restarted to background color), simply reused as it was left from previous calculations causing multiple images to be successively built up, or reset by unconditionally overwriting one of the other pixel buffers, or conditionally overwritten by several of the pixel buffers (using brute force 'Z' comparisons), effectively merging the 3-D surfaces together. The interactive advantages of resetting and merging pixel buffers together can be shown in an animation example: A bird could be interactively flown around in a forest by precalculating a view of all the trees and storing the 2-D surfaces in a pixel buffer for a background image a bird could be interactively animated with a very quick response time between frames, because only the bird would need to be recalculated each frame. Yet the bird would be subject to all the same 3-D cues as if the trees were also being recalculated. The bird could fly around the trees, disappearing as it goes behind them, or it could fly through grass or leaves, becoming only partially obscurred. Further, just as in conventional animation, the background landscape of trees and mountains could be precalculated much larger than the actual 512 x 512 TV viewing resolution so that the field of view could be moved around the scene, only showing part of it at a time. Using this technique, the animator can enjoy the benefits of animating with complex imagery without losing the benefits of interactiveness as a result of slow response time.

A variation of merging pixel buffers is to store only as much of the pixel buffers as needed to contain a single object (or object group). Using this technique an animator can not only interactively manipulate an object around a complex 3-D background, but can now manipulate parts of the background also. This is done by calculating an object at 512 x 512 but only using a pixel buffer just big enough to hold it (such as 136 x 120 or 250 x 74, etc.). Then for each frame, these pixel buffer sections can be interactively moved around with a very rapid response time while benefiting from 3-D cues like overlap and intersection. Precalculating several views of an object and storing them in this manner also has advantages for object instancing and high complexity data generation techniques in which the user can interactively work on part of a detailed object while the rest of the object has been precalculated.

SOME PROBLEMS AND POSSIBLE SOLUTIONS

One problem in our approach is that all surfaces are treated as visible, so that a fair amount of processing is done to each polygon whether it is ultimately visible or not. In a low complexity environment this problem has generally been solved by not using a frame buffer, but instead depth sorting all the polygons and finding the visible surfaces using segment overlapping or image subdivision techniques. The problem with this approach is that with a very complex scene, such as a view of a city, the time spent sorting can easily be far longer than the time spent using a brute force approach. A more acceptable way of modeling the city example is to use a hierarchical approach such as Clark (11) used. However this method also suffers from non-linear sorting/searching steps. A simple linear solution does exist using a pixel buffer, but requiring extra bits per pixel. Using this approach the city example could exist as a very simple block model and as a very detailed (windows, doors, interiors, etc.) model. The simple model would be scanned first, using a brute force algorithm to find visible surfaces and storing an object number with every pixel indicating where it came from. Then the pixel buffer could be searched, making up a list of objects which had some visible pixels in the rouch scan. Then the detailed model could be scanned but with only those objects which showed up on the visible list. Thus for a small preprocessing cost, a huge number of polygons can be eliminated.

Another problem with using a pixel buffer is that only two surfaces can be involved when pixels are being modified. Many situations, notably scanning a mixture of transparent and solid surfaces and using the anti-aliasing techniques of Catmull and Crow, require not only three or more surfaces to interact but also need an ordering of the surfaces from back to front. In an environment where the production of realistically simulated still photographs are desired, there is no easy solution to this problem. However in an animation environment in which simplicity and speed are more important than accurately simulating a physical phenomenon, we believe this problem can be minimized so as to produce acceptable results. In the case of transparency, a lot of the problems with mixing solid and transparent surfaces can be avoided by scanning all solid objects first, and then transparent ones. This avoids the problem that occurs when a solid pixel lies between two transparent pixels that have already interacted and modified each other. The problems of trying to modify a set of randomly positioned colored transparent surfaces can be minimized by keeping a separate color value at each pixel for a transparent pixel which lies in front of a solid pixel or background. This, plus keeping transmittance bits and extra bits indicating whether a pixel is solid or transparent and whether it's been modified or not (and by how much), will result in an image that is merely a simulation of transparency, but which will be consistent and produce an adequate effect.

Solutions to the aliasing problem require a great deal of computer time making a software implementation impractical for highly complex data. It can take two to five times longer than the visible surface calculation. We plan to implement, partially in hardware, a special case solution to this problem. Our scheme involves calculating pictures at 1024 x 1024 resolution and averaging the intensities (not colors) of four pixels into one pixel (using dominant chroma) into a normal 512 x 512. We also plan to experiment with various low pass filters implementing them in microcode.

TEXTURE

In our attempt to handle visually complex objects, we are concerned not only with objects possessing hundreds of thousands of polygons to define smooth surfaces but also with the processing necessary to give the surface of an object the appearance of a particular texture. By "texture" we mean the visual properties associated with a surface when that surface is made of a certain material, e.g., plastic, wool, plaster, hair, etc.

To date, there have only been a few significant attempts known to us at accomplishing this. The most recent, and most successful, was presented at SIGGRAPH '78 by Blinn. Blinn's paper describes a technique using a quadrilateral patch defining a normal-vector perturbation function. This patch is stretched to fit each surface patch and new normal vectors are computed at each surface point on the patch. Although very effective in certain situations and as a demonstration, the technique has three major drawbacks. First, because the texture patch is repetitively mapped onto every surface patch in a uniform manner, it is subject to forming undesirable patterns on the object when the actual texture possesses no such pattern (due to the pattern formed by the surface patches themselves, such as in a crater texture mapped onto a sphere consisting of eight uniform surface patches). On the other hand, the technique is also capable of destroying an overall pattern of the texture if the surface patches are not appropriately aligned (as in the case of mapping a brick texture onto an irregularly shaped object).

Second, because the visible surface portion of the display processing is performed without regard to the texture, the image calculated when using certain textures will not appear to be "reasonable." Not only will a silhouette of an object not reflect the texture on the surface (e.g., bumps, bricks, hair), but relatively deep textures will not hide adjacent features (e.g., deep holes, craters, mountains).

Third, because of the simplification used in calculating the perturbed normal vector (i.e., that the value of the bump function is negligably small) in order to make the calculation tractable, patterns with relatively large texture features (e.g., long hair, deep depressions, large bumps) are not correctly calculated.

We have taken a different approach to representing a textured object in that we define a three-dimensional perturbation of the object's surface (i.e., surface detail) as well as variations in reflectance and color properties along the surface. Because our high complexity data consists of hundreds of thousands of small polygons we can rearrange their position as well as assign individual color and reflectivity in order to more realistically model "physical" texture on an object. Instead of simulating the reflectivity of craters on the surface of an object (a la Blinn) we can actually "create" craters on the surface. Our efforts up to this point in time have been a relatively simple repetitive replacement of a physical texture onto an arbitrary solid of revolution.

To specify a texture, the user interactively selects the rectangular size of the patch, and the displacement, color and reflectivity for each point on the rectangular patch. The size of the patch is controlled by two dials. The user selects a point on the patch using two additional dials. The user specifies the displacement, color and reflectivity by pressing the appropriate function button and adjusting the input-value dial. At all times the texture rectangle is displayed on the TV monitor with each point's value of the attribute the user is currently concerned with encoded in color. Thus the user can create, modify or view the attribute values of a texture. The user can also request a visible surface display, with a modeled light source, of a texture patch on the monitor. Additionally he can request the display of the texture on a primitive shape, for example, a sphere or cube.

In order to specify a textured object, the user selects a contour line and a texture patch (or makes hi own). At the time the object is to be displayed, a routine is called which generates the solid of revolution defined by the contour line while simultaneously applying the perturbation function to the surface being generated. This procedurally based object definition results in a flexible and straightforward representation as well as a great savings in space allocated to such representations.

In our current implementation of processing texture, the texture specification contains the number of object slice points to be replaced as a group (call it k) and an n x m array of x, y, z displacements. As each object slice is generated (by rotation of the object contour line) each k points of the slice are replaced by the n points of the appropriate column of the texture array. The number of the appropriate column is the number of the object slice modulo m+1. The replacement is performed by using the first and kth points of the group to be replaced to determine the rotation, scale and translation operations necessary for the first and nth points of the texture column to coincide with the positions of the first and kth points of the object group. The n transformed texture column points are then substituted for the k points of the untextured object. This process is repeated for each successive group of k points in the object slice. When the next object slice is generated, the next column of the texture array is used for the replacement.

Once we have a good understanding of the results obtainable by our method, we hope to develop techniques by which we can impart a texture onto an irregularly shaped object. This is a difficult problem in the case where the texture has some overall pattern and this pattern must be kept intact over the surface of the object. In order to avoid some of the problems cited with regard to Blinn's technique it may be necessary to adopt other methods of imparting a texture on an object. For example, a texture may be propagated along the surface of an object spreading out from an arbitrary starting point. Or each vertex may be considered a probabilistic finite state automation which will stabilize into a state dependent on the texture specification, chance, and the state of the neighbors of the vertex. Another, although simpler, extension to our texture specification is the introduction of randomness. This is very desirable for use in many textures such as hair, craters and plaster. This could be accomplished either by successively selecting from a number of texture patches at random or by random modifications to a standard texture patch.

SMOKE

As Blinn says in the summary of his dissertation, existing display techniques leave much to be desired when it comes to the display of "soft" objects such as clouds, fur, smoke, and fire. The CGRG has been investigating an alternative technique, which allows for the possibility of representing solid or nonsolid objects, not as a collection of boundary curves (patches, polygons, etc.), but rather as a collection of points. Each of the points comprising the object can be treated as a separate data entity, and will have associated with it properties of intensity and



Smoke Cloud 300,000 points chromaticity, position, and orientation, as well as any properties necessary for animation purposes.

The main motivation for this investigation is that non-solid objects can be more accurately represented, with more realistic visual cues. such as the billowing of clouds, the dancing of fire, or the flowing of water. Another desirable property of a point representation is that a solid object could indeed be solid in its representation, as all points inside the boundary surfaces could be defined as well as those on the boundaries. Some data input techniques are pointsampling techniques, such as medical cat-scans, and raster scan output displays can be considered as a collection of ordered points (pixels). Consequently, determination of visible surfaces reduces to a Z comparison of points in the data structure.

There are certainly many disadvantages to this approach, such as the large number of points necessary to describe an object, and the need to somehow relate a point to it's nearest neighbors. But we believe that a technique of this type is important in achieving the degree of realism desired in an animation environment.

We have experimented with a point-based object representation by creating an image of a cloud of smoke rising from an elevated source. This first attempt uses the theory of particular clouds attributed to Green and Lane (11) as the basis for generating the data. The intensity of each point (x,y,z) in the cloud is dependent on the density of the particles there. This can be determined using the equation for elevated sources diffusion:

$(x,y,z) = \frac{Q}{C_y C_z \overline{u} x^{2-n}} \exp$	$-\frac{y^2}{c_y^2 x^{2-n}}$	exp	$\frac{-(z-h)^2}{c_z^2 x^{2-n}}$	+ exp	$\frac{-(z+h)^2}{c_z^2 x^{2-n}}$
---	------------------------------	-----	----------------------------------	-------	----------------------------------

where

X = is concentration in g/cm³ at pt (x,y,z)

Q = is rate of emission

h = is height of source

and C_v and C_z are temperature gradients determined from wind velocity, turbulence, dif-fusing power, viscosity of air, and eddy velocity.

By varying one or more of the parameters in the model, different images of the clouds can be generated, and these can be combined to create an image that more closely approximates the actual form and intensity of experimental smoke clouds.

EXPERIMENTAL RESULTS

The animation subsystem implemented on the PDP-11/45 was intended as an experiment to better understand the problems associated with highly complex data. No attempt was made to optimize code for greater speed and efficiency. In this system it takes the display algorithm approximately four and one half minutes ("wall" time) to process and display 150,000 triangles or 450,000 edges. We are now designing a new system for our VAX/780 and new graphics display processor. While it is difficult and dangerous to predict the expected improvement in performance, we hope that the new system will be a significantly better one.



Tribbles - 500,000 triangles



Burley, Idaho - 25,000 triangles



Beldar's Hat - 12,000 triangles



Cocklebur - 46,000 triangles

GRAPHICS DISPLAY PROCESSOR

The VAX 11/780 Graphics System is a hardware/ software interface designed to aid in the creation, calculation, and output phases of computer animation. The main objective behind the hardware/software design is to reduce the time between object description and final animation output while at the same time increasing the artists' control over the process.

The graphics system consists of three main parts. They are the Z buffer, the 2-D color frame buffer(s), and the run length animation port (see Figure 2). The purpose of the Z buffer is to take in 3-D run length formatted data and to produce, on a frame by frame basis, 2-D run length data which describes the visible surfaces of the original 3-D information. The Z-buffer sorting algorithm is implemented by a bit-slice microprocessor because of the relative ease with which the algorithm may be modified.

For example, adding anti-aliasing or transparency processing requires making changes to the microprocessor's writeable control store (microprogram). This approach also gives the capability to process a higher resolution 1024×1024 image space and to do intensity averaging in microcode.

Operation of the algorithm involves translating the 3-D run length data into a pixel format. Each generated pixel is Z value sorted into the core memory. After all the runs for a frame have been processed, additional processing (such as low pass filtering) may occur. Then the pixel representation of the image is translated into the 2-D format that the run-length port recognizes and sent back to the VAX 11/780 for storage.

The purpose of the 2-D color frame buffer(s) is to provide a static 512×512 pixel image. These will be used for operator interaction during the object definition phase.



The run-length port is the animation port of the system. It takes in 2-D run-length formatted data and produces real-time NTSC video suitable for tape recording. The run-length port accepts 2-D run-lengths data from the VAX DMA interface and translates this data into the appropriate real-time NTSC video.

The inclusion of the core memory in the data path also allows for the playback of multiple frame (less than 6 metabytes in length) sequences of high complexity. These sequences are of such high data band widths that the DMA interface might not be able to supply data fast enough.

CONCLUDING REMARKS

The details of the graphics language for ANTTS is the subject of another paper but it is important to note some of its basic capabilities. It can handle various arithmetic and transformational commands which are either unscheduled or scheduled (of a type similar to those of ANIMA II) as well as incorporating commands used to modify the flow of control. Variables can be set either programmatically or interactively which allows great flexibility in controlling the animation.

BIBLIOGRAPHY

Blinn, J. F., Simulation of Wrinkled Surfaces, SIGGRAPH, 1978.

Brooks, J., et al., An Extension of the Combinatorial Geometry Technique. NTIS Report AD-782-883, August, 1974.

Catmull, E. A., A Subdivision Algorithm for Computer Display of Curved Surfaces, Univ. of Utah, UTEC-CSC-74-133, Dec., 1974.

Clark, J. H., Hierarchical Geometric Models for Visible Surface Algorithm, CACM, October, 1976.

Crow, F. C., The Aliasing Problem in Computer Synthesized Shaded Images, UTEC-CSC-76-015, Dept. of Computer Science, Univ. of Utah, 1976.

Hackathorn, R., The ANIMA II System, SIGGRAPH '77.

Jones, B., An Extended ALGOL-6, for Shaded Computer Graphics, Proc. ACM Symp., 1976.

Myers, A. J., "An Efficient Visible Surface Program," Grant No. DCR 74-007'68A01, Tech. Rep., National Science Foundation, 1975.

Newell, M. E., The Utilization of Procedure Models in Digital Image Synthesis, Computer Science, Univ. of Utah, UTEC-CSC-76-218, 1975.

Staudhammer, J., and Eastman, J. R., Computer Display on Colored Three-Dimensional Object Images. Proc. Annu. Symp. Comput. Arch., 1975.

Sutherland, I. E., Sproull, R. F., and Schumacker, R. A., A Characterization of Ten Hidden-Surface Algorithms, Comput. Surv., 1974.

Wein, M., and Burtnyk, N., A Computer Animation System for the Animator, UAIDE, 1971.